リアルタイムOSのテスト設計と検証の事例
ー無限に存在するテストケースを有効に絞り込み問題を発見した例ー

田村 聡† 酒井 由夫‡
†イー・フォース株式会社
〒103-0006東京都中央区日本橋富沢町5-4
‡NPO法人組込みソフトウェア管理者・技術者育成研究会
〒103-0011東京都中央区日本橋大伝馬町6-7

eForce

アジェンダ

1. はじめに
2. リアルタイムOSとは
3. リアルタイムOSのテスト設計
4. リアルタイムOSのテストで発見した問題と考察
5. ミューテックス部分で発見した問題と考察
6. まとめ

はじめに

μITRON4.0仕様に準拠した２種類のリアルタイムOSのテスト事例を紹介する

RTOS特徴

- ◆リアルタイムOSは，イベントドリブンのOS
- ◆イベント発生を起因として，タスクスケジューリングを実施

- ➢ リスクベースのアプローチからテスト仕様を設計
- ➢ スケジューリング機能を考慮したテスト仕様を設計
- ➢ 作成されたテスト仕様をもとに，テストフローを設計
- ➢ 作成されたテストフローをもとに，テストプログ

アジェンダ

1. はじめに
2. リアルタイムOSとは
3. リアルタイムOSのテスト設計
4. リアルタイムOSのテストで発見した問題と考察
5. ミューテックス部分で発見した問題と考察
6. まとめ

リアルタイムOSとは

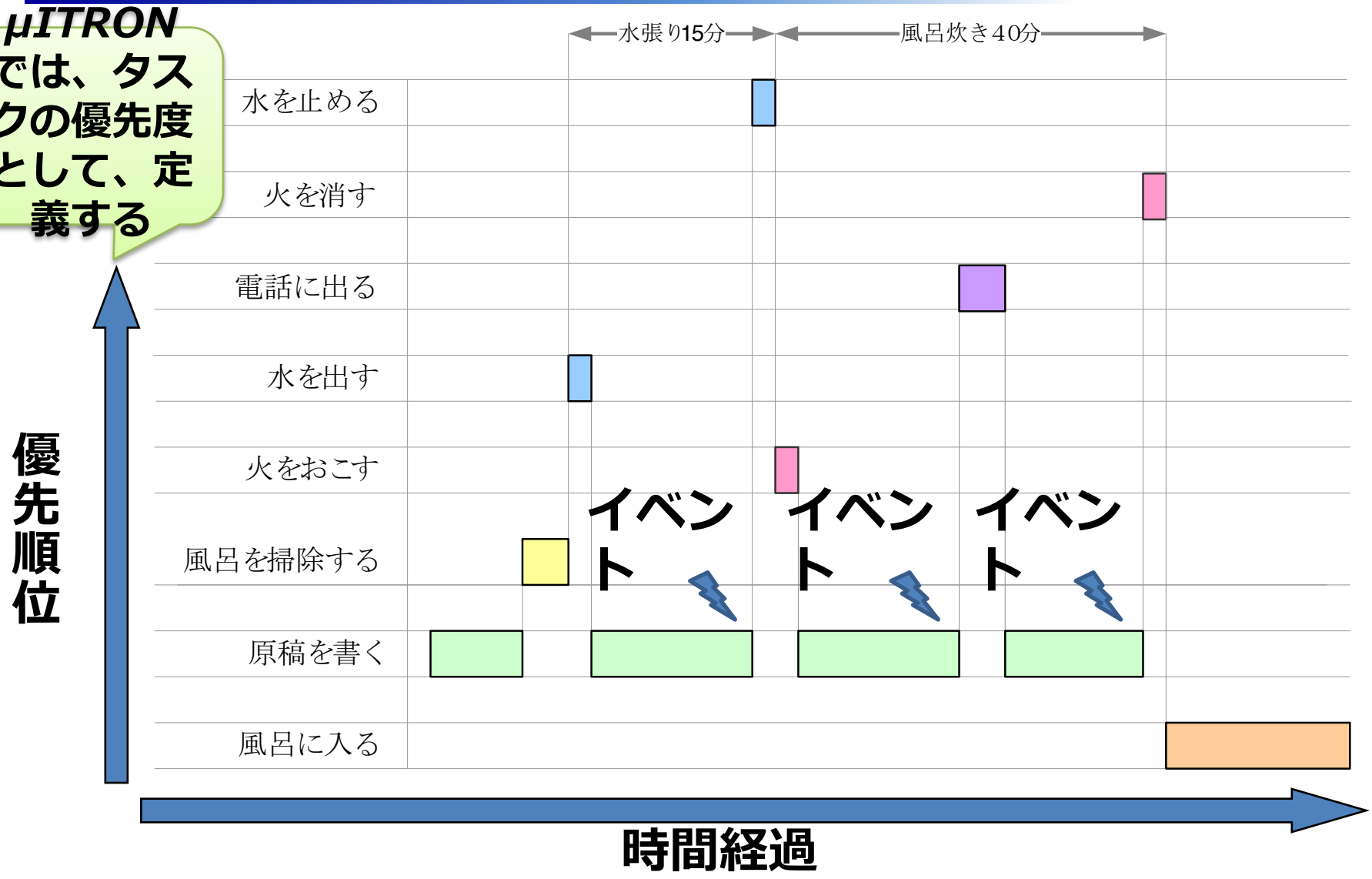
◆昔の風呂炊きの作業をリアルタイムシステムとして考えてみる

- 五右衛門風呂を沸かす手順(仕様)
  1. 風呂釜を洗い栓をして水を注ぐ
  2. 約15分たったら水を止める
  3. 水がいっぱいになったら薪を燃やす
  4. 約10分たったら湯加減を見てちょうど

【ポイント】
- 複数の作業を自分一人で実施する
- 時間を含む限られた資源を有効に使いたい
- 水を止めそびれると溢れてしまう
- 火を消し忘れると火事になるかもしれない
- 水を入れているとき、風呂を沸かしている間は他のことをしていてもよい

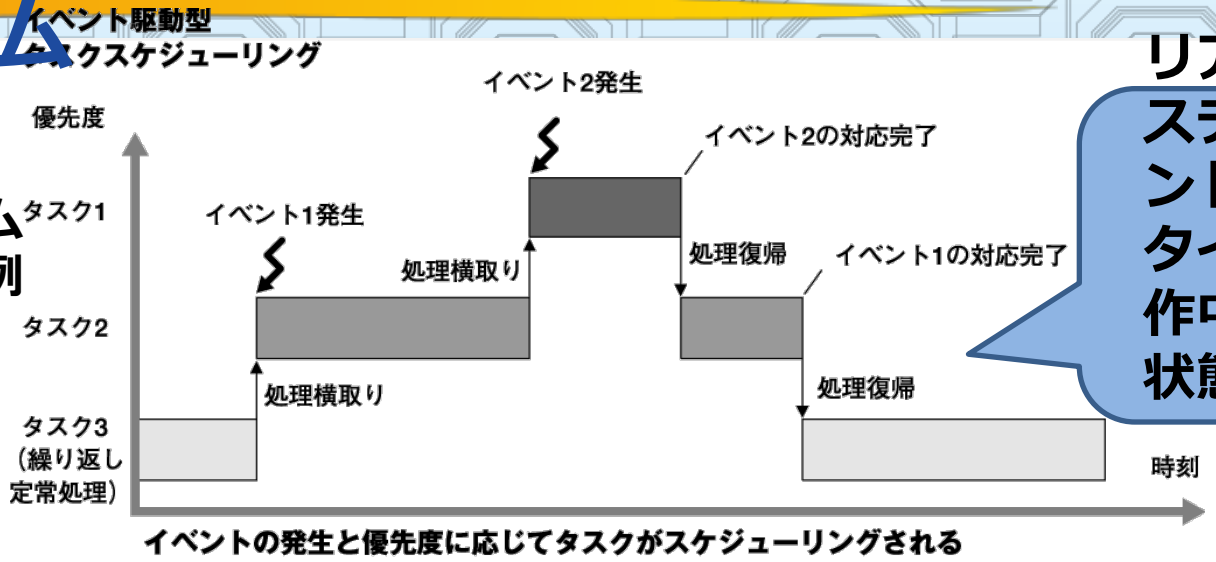
五右衛門風呂を沸かす手順(スケジューリング)

μITRONでは、タスクの優先度として、定義する



リアルタイムシステムとタイムシェアリングシステム

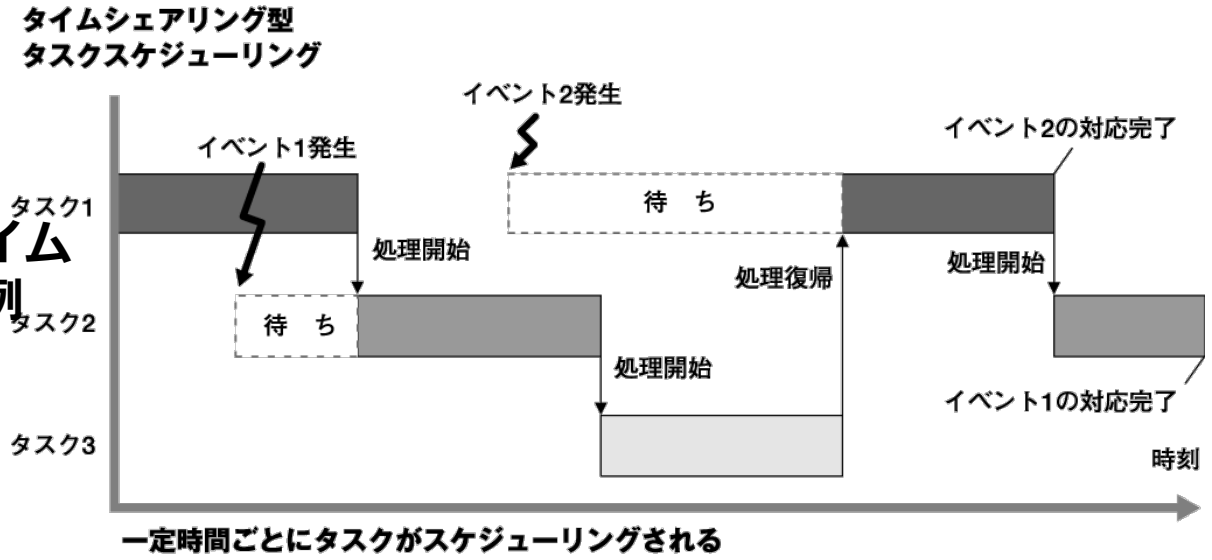
リアルタイムシステムの例



イベントの発生と優先度に応じてタスクがスケジューリングされる

リアルタイムシステムではイベントが発生したタイミングで動作中のタスクの状態を変更・管理が必要

非リアルタイムシステムの例



一定時間ごとにタスクがスケジューリングされる

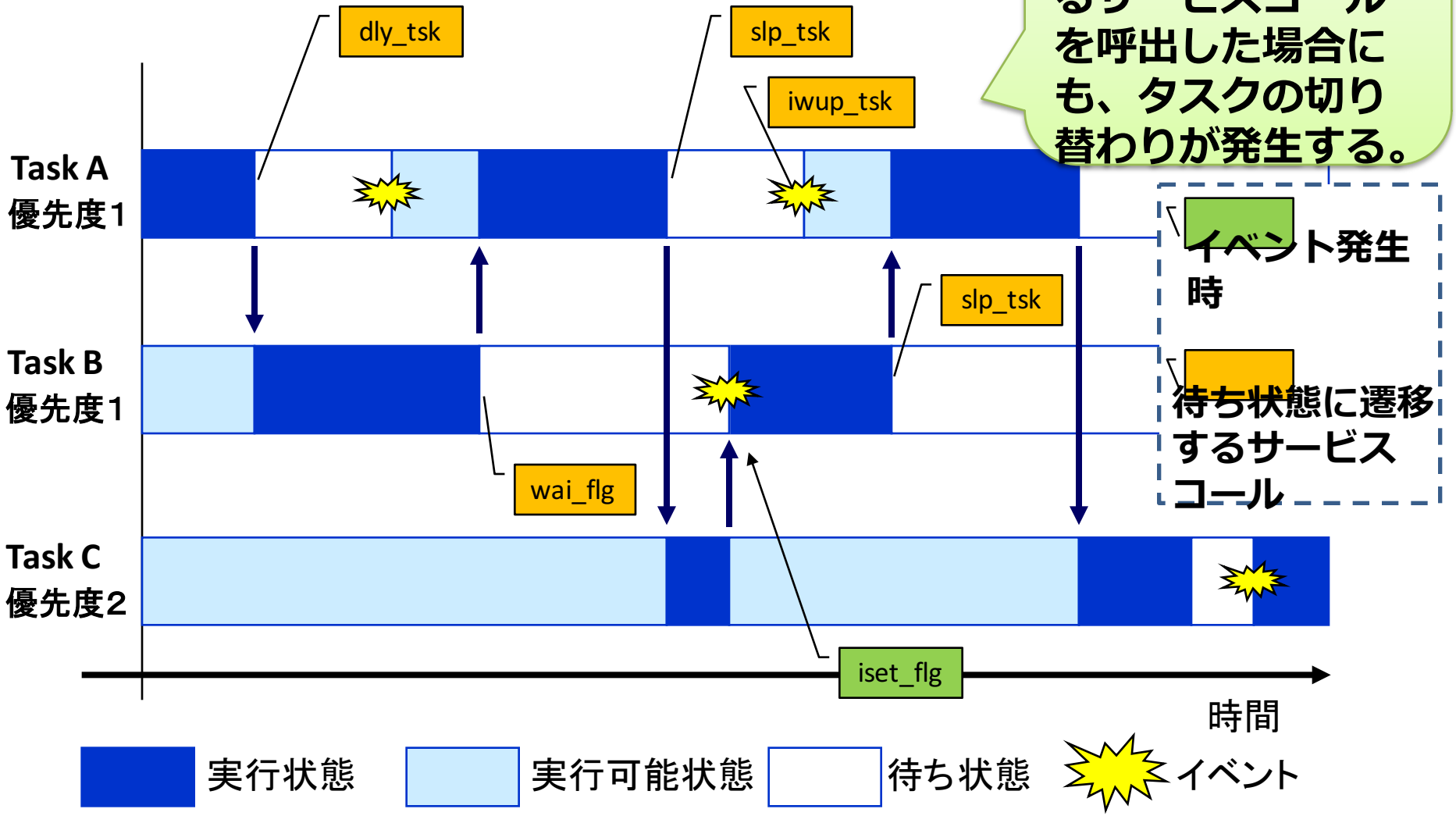
※T-Engineフォーラムの資料を基に作成

アジェンダ

1. はじめに
2. リアルタイムOSとは
3. リアルタイムOSのテスト設計
4. リアルタイムOSのテストで発見した問題と考察
5. ミューテックス部分で発見した問題と考察
6. まとめ

タスクスケジューリング

タスクスケジューリングの例

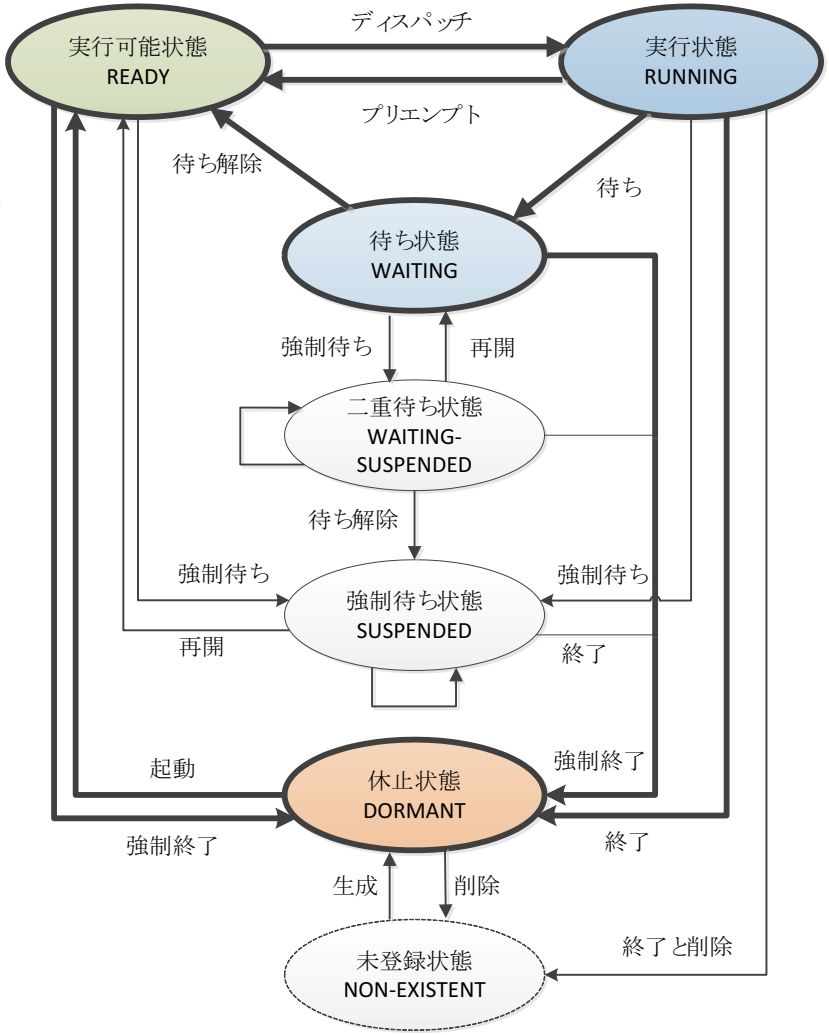


イベント発生以外にも、*µITRON*の待ち状態に遷移するサービスコールを呼出した場合にも、タスクの切り替わりが発生する。

タスクの状態遷移とテストで必要なこと

RTOSのタスクは，他のタスクや、割込みから呼び出されるサービスコールによって状態遷移する。

- 単純にRTOSのシステムコール（サービスコール）を呼んで，戻り値が期待通りかどうかを確認するというテストではRTOSのテストは完結しない。
- OSの処理中にも、割込みが介入する可能性もあるため，C言語のソースコードレベルでカバレッジを100%にしても完全なテストを実施したとは言えない。



μITRONのタスクの状態遷移

RTOSのテスト設計の指針

- タスクの状態とイベントの種類や発生タイミングによってテストケースが多岐に存在するため，割込み発生のタイミングを含めた，すべてのケースを網羅したテストは難しいという前提でテストを設計

【*RTOS*のテスト設計の指針】

- *RTOS*利用者の利用環境を想定して，利用される可能性の高いテストシナリオ（想定するタスクと発生させるイベント）を策定する。
- 特に漏れが生じやすい複雑な仕様のサービスコール（ミューテックスなど）のテストを重点的に実施する。
- *RTOS*テストの評価者（*RTOS*の利用者）がテスト結果を適切に評価できるように，テスト内容とテスト結果，またテストを実施した*RTOS*のバージョンとテスト環境を

利用者のリスク低減を目的としたリスクベースアプローチでテスト設計をすることが重要

RTOSのテスト仕様作成

- ユーザーズガイド（RTOSの機能仕様）を元に，タスクやイベントを想定し，テスト仕様を作成する

- エラー条件テスト
  - 各エラーコードになる条件を検討
  - さらに呼出しコンテキストの条件で分類
- 機能テスト(解説からの検討)
  - *tskidにTSK_SELFを指定する*
    - *tskpriにTPRI_INIを指定する*
    - *tskpriにTPRI_INI以外を指定する*
  - *tskidにTSK_SELF以外を指定する*
    - *tskpriにTPRI_INIを指定する*
    - *tskpriにTPRI_INI以外を指定する*

次ページに続く

【書式】

ER ercd = chg_pri (ID tskid, PRI tskpri) ;

【パラメータ】

ID	tskid	変更対象のタスクの ID 番号
PRI	tskpri	変更後のベース優先度

【エラーコード】

E_ID	不正 ID 番号（tskid が不正、あるいは使用できない）
E_NOSPT	対象タスクが制約タスク属性
E_PAR	パラメータエラー（tskpri が不正）
E_OBJ	オブジェクト状態エラー（対象タスクが休止状態）

【呼び出しコンテキスト】

タスク	可
タイムイベントハンドラ	可
割込みサービスルーチン	不可

【解説】

tskid で指定されるタスクの現在優先度を、tskpri で指定される値に変更します。 tskid には、コンフィグレータで生成した際のタスク ID の定義名を使って指定します。tskid に TSK_SELF（=0）が指定されると、自タスクを対象タスクとします。また, tskpri に TPRI_INI（=0）が指定されると、対象タスクの現在優先度を、タスクの起動時優先度に変更します。

対象タスクが実行できる状態である場合、タスクの優先順位を、変更後の優先度にしたがって変化させます。変更後の優先度と同じ優先度を持つタスクの間では、対象タスクの優先順位を最低にします。

ユーザーズガイドの抜粋

RTOSのテスト仕様作成

- 複数のタスク，タイムイベントハンドラ，割り込みイベントを想定

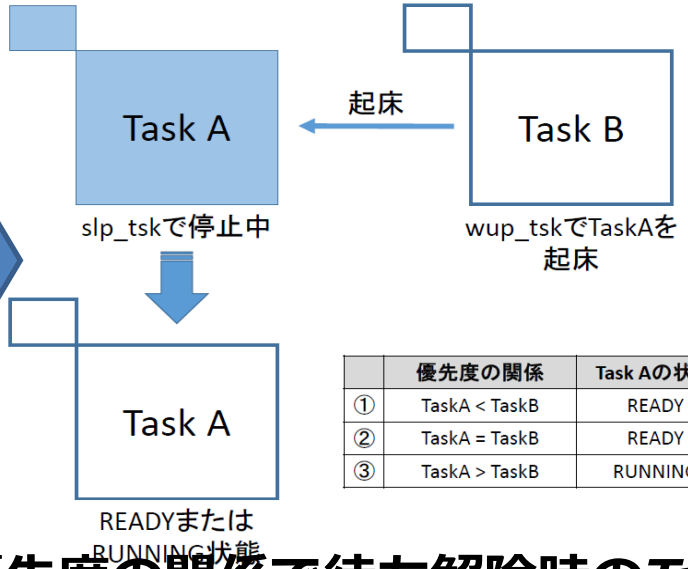
- 機能テスト(続き)
  - *tskidにTSK_SELF以外を指定する*

機能テスト タスク	B	tskid に TSK_SELF(=0) を指定し、tskpri に TPRI_INI(=0)を指定した場合、自タスクのベース優先度が起動時の優先度数に変更でき、戻り値がE_OKとなること	---	---	---
	B.1	変更後のベース優先度を変更前よりも高くした場合、	Task016	21-23	
	B.2	変更後のベース優先度を変更前よりも低くし、変更後の優先度数と同じ優先度の実行可能状態のタスクがある場合、この時自タスクは[illegible]尾につながれ、レディキューのパッチする	Task016	25-29	
	B.3	変更後のベース優先度を変更前よりも低くし、変更後よりも優先度の高いタスクが実行可能状態にある場合、自タスクは実行可能状態となること	Task016	30-36	

chg_priの例

テスト仕様策定例

- この場合，タスクの状態遷移を考慮し，待ち状態の要因(セマフォなど，OSの他のオブジェクト)も考慮する
- 右に示す，タスクの優先度の関係を利用して，[illegible]ューリングを考慮する
- テスト仕様策定例に抜粋を記載する
- タスクコンテキストでは，59個の条件
- タイムイベントハンドラでは，[illegible]25個の条件

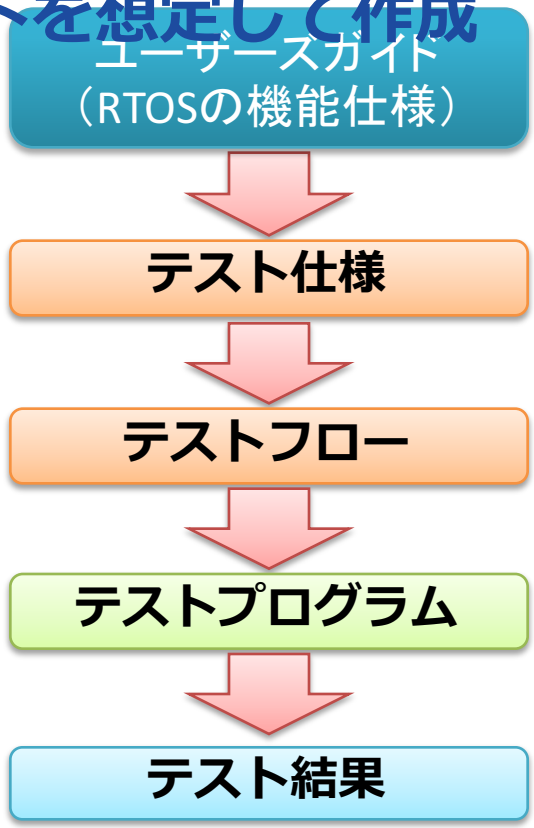


	優先度の関係	Task Aの状態
①	TaskA < TaskB	READY
②	TaskA = TaskB	READY
③	TaskA > TaskB	RUNNING

優先度の関係で待ち解除時の*Task A* と *TaskB* の状態が変化

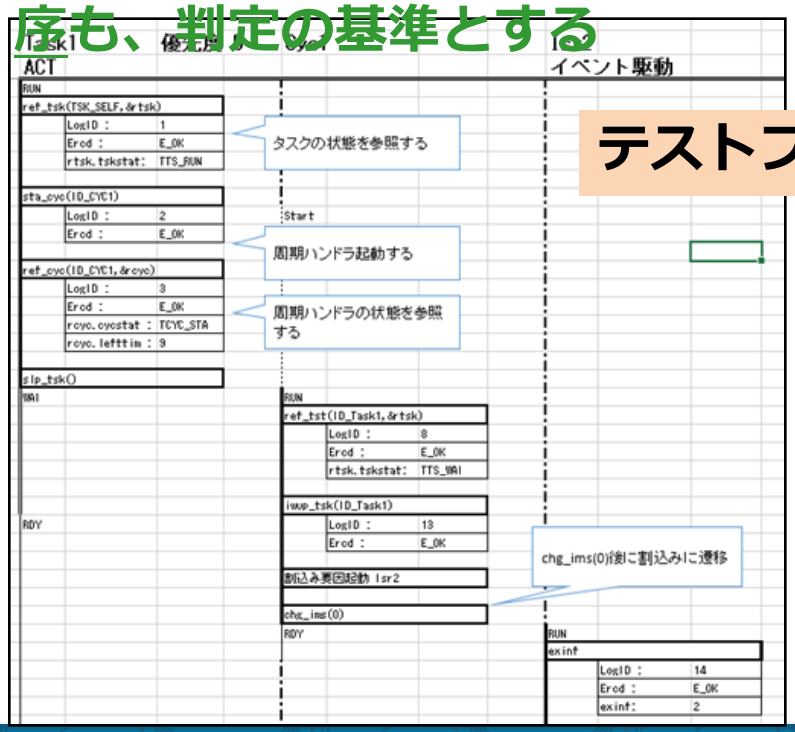
RTOSのテストフロー作成

- 複数のタスク，タイムイベントハンドラ，割り込みイベントを想定して作成



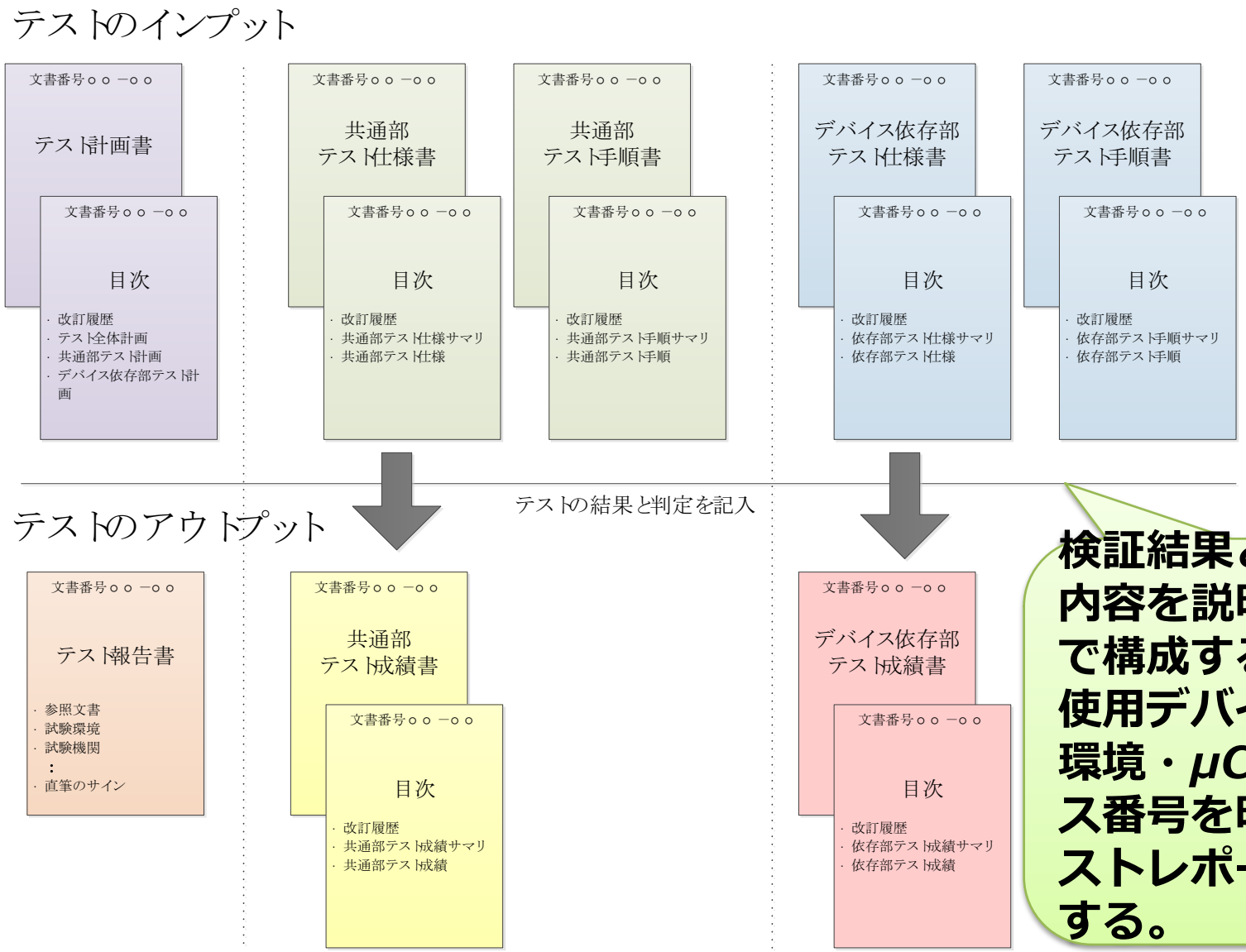
テスト設計と検証の流れ

- テスト仕様
  - エラー条件テスト，機能テストを作成
- テストフロー
  - テスト仕様の各項目を検査するためのフローを作成
  - 複数のタスク・タイムイベントハンドラ・割込みハンドラ環境でのフローを作成
  - テストフローでは、設定値・戻り値・処理順序も、判定の基準とする



テストフロー設計例

RTOSのテストのインプットとアウトプット（レポート）



検証結果と、検証の内容を説明した資料で構成する。
使用デバイス・開発環境・*μC3*のリリース番号を明記したテストレポートを作成する。

アジェンダ

1. はじめに
2. リアルタイムOSとは
3. リアルタイムOSのテスト設計
4. リアルタイムOSのテストで発見した問題と考察
5. ミューテックス部分で発見した問題と考察
6. まとめ

RTOS（μC3）を検証した際のテスト工数とテスト結果の分量

種別	検証したサービスコール数	テスト設計時間（のべ）	検証記録頁（A4）
Compact	64※1	約110日	約2030頁
Standard	129※1	約160日※2	約5400頁

※1. ixxx_xxxのサービスコールは，数に含まない
※2. Compactの設計資料を一部利用

コンパクトなRTOSでも多大なテスト設計時間が必要であり，検証記録も膨大な量となる。また，CPUやその他の使用環境が変われば，再テストも必要となる。
→したがって，設計したテスト環境やテストシナリオは貴重な資産であり，テストは再現できるようにする。また，結果の集計等では、判定ミスを防ぐためにも、マクロ等を利用して，自動で実行する。

μC3/Compactの検証で見つかった問題

No.	概要	見落としの原因
1	優先度変更のサービスコール*chg_pri* で優先度の変更が変化しない。	*chg_pri*の呼び出しで，タスク優先度設定の想定漏れ。
2	周期ハンドラ内でその周期ハンドラの動作を停止する *stp_cyc*を呼び出すまでの区間に，周期ハンドラや時間監視付きサービスコールの時間待ちを解除した場合，周期ハンドラ終了後に暴走する。	周期ハンドラ内で，時間待ちの処理を解除するテストの想定漏れ。
3	周期ハンドラからタスクの状態参照 *ref_tsk*を呼び出し，当該タスクが時間待ち状態からタイムアウトする時刻と一致していた場合，タスクの状態としてタイムアウトするまでの時間が正しくない。	*ref_tsk*の呼び出しで，境界条件のテスト漏れ，また，システム状態の想定漏れ。
4	チック時間で割り切れない起動周期を持つ周期ハンドラの状態参照 *ref_cyc*を呼び出した場合，周期ハンドラを次に起動する時刻までの時間*lefttim*が*1*チック時間だけ短くなる。	*ref_cyc*の呼び出しで，周期ハンドラの起動周期と，チック時間の関係の想定漏れ。

μC3/Compactの検証で見つかった問題の考察

- *No.1*
  - *chg_pri*は，指定されたタスクの優先度を変更する処理であるが，変更後の優先度値として，－値を指定した場合に不具合が発生した。
  - *μITRON*では，優先度値として，符号付き整数であることから，境界値のエラー判定が不足していた。
- *No.2*
  - 周期ハンドラ中に，同じ時刻にタイムアウトする時間待ちの処理(*txxx_xxx*サービスコール，*dly_tsk*サービスコール，周期ハンドラ)について，*rel_wai*というサービスコールで待ち状態を強制解除すると，発生した事象。
  - 周期ハンドラ処理中に，同じ時刻の待ち解除という条件と，強制待ち解除という複数の条件が重なった場合などへの，条件考慮が不足していた。
- *No.3, No.4*
  - 周期ハンドラ処理中の，時間判定の考慮が不足していたため，発生した事象。

μC3/Standardの検証で見つかった問題

No.	概要	見落としの原因
1	時間監視付きサービスコール（txxx_yyy）のtmoutに指定できる最小値TMO_FEVR(=-1)を超えた値(-2以下)を指定しても，戻り値にE_PARが返らない。	時間監視付きサービスコール（txxx_yyy）での，境界条件のテスト漏れ。
2	TA_TPRI | TA_WSGL 属性が指定されたイベントフラグに対して(i)set_flgサービスコール発行後，ref_flgサービスコールを発行すると，ref_flg内にて誤ったアドレスに不正アクセスする。	設定条件(TA_TPRI |TA_WSGL)を想定した場合のテスト漏れ。
3	周期ハンドラ内からID番号に自IDを指定してref_cycサービスコールを発行すると，cycstatに誤った動作状態が返る。	ref_cycの呼び出しを，自周期ハンドラに対しての実施することの想定漏れ。
4	タスクコンテキストからID番号にTSK_SELF(=0)を指定してact_tskサービスコールを発行すると，戻り値にE_IDが返る。	act_tskの呼び出しを，自タスクに対しての実施することの想定漏れ。
5	chg_priサービスコールのtskpriに指定できる最小値TPRI_INI(=0)を超えた値(-1以下)を指定しても戻り値にE_PARが返らない。	chg_pri呼び出しでの，境界条件のテスト漏れ。
6	ivsig_ovrサービスコール発行時，SSB(システムサービスブロック)が不足した場合,戻り値にE_NOMEMが返らない。	割込み処理内で，管理パラメータの設定不足時のテスト漏れ。
7	既にオーバーランハンドラが定義されている状態で，def_ovrサービスコールを発行してオーバーランハンドラを再定義する場合，再定義できない	オーバーランハンドラを再定義した場合の想定漏れ。

μC3/Standardの検証で見つかった問題

No.	概要	見落としの原因
8	ミューテックスをロックしているがタスク終了する時に，そのタスク以外でロックしているミューテックスのロックが解除される。	ミューテックスでは，タスクの終了時に，そのタスクがロックをしているミューテックスのロックを解除する。その時，他のミューテックスへの影響の想定漏れ。※3
9	優先度継承プロトコル(TA_INHERIT)で一度優先度を継承し，そのミューテックスのロックを解除後，2回目以降，優先度が継承されない。	ミューテックス動作の詳細仕様の策定及びテスト漏れ。※3
10	優先度継承プロトコル(TA_INHERIT)で優先度継承後，そのミューテックスのロックを解除しても，優先度が元に戻らない。	ミューテックス動作の詳細仕様の策定及びテスト漏れ。※3
11	優先度継承プロトコル(TA_INHERIT)でloc_mtxサービスコール発行後，ロックできず値にE_TMOUTとなった場合，優先度が元に戻らない。	優先度継承プロトコルで，優先度継承により変更されたタスク優先度を戻す処理の想定漏れ。※3
12	chg_imsサービスコールのimaskに1を指定した場合，loc_cpuサービスコールと動作が一致しない。chg_imsサービスコールのimaskに0を指定した場合，unl_cpuサービスコールと動作が一致しない。	特定のデバイスシリーズで，コンテキストの想定漏れと，タスクの切り替え条件の想定漏れ。
13	アラームハンドラ内からID番号に自IDを指定してref_almサービスコールを発行すると，almstatに誤った動作状態が返る。	ref_almの呼び出しを，自アラームハンドラに対しての実施することの想定漏れ。
14	tget_mplサービスコールにて可変長メモリブロックの獲得を待っているタスクが，タイムアウトにより待ち解除後，新たに待ち行列の先頭となったタスクが，メモリブロックを獲得可能にも関わらず獲得できない。	可変長メモリブロックの獲得判定処理での想定漏れ。

μC3/Standardの検証で見つかった問題の考察

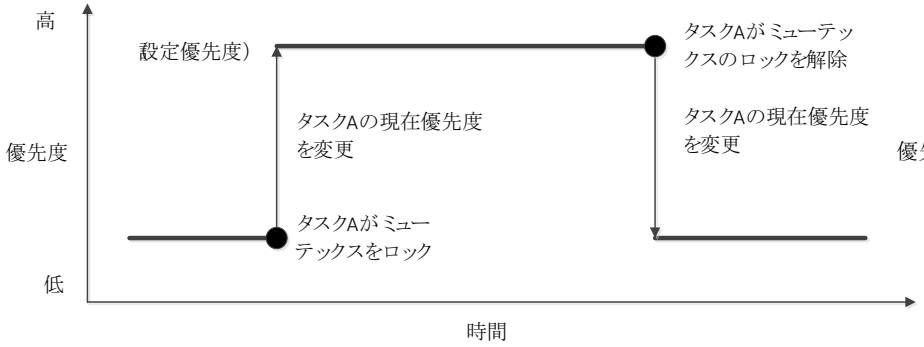
- *No.1, No.5*
  - 境界値エラー判定のテスト不足。
- *No.2*
  - 設定条件(*TA_TPRI |TA_WSGL*)の設定条件は、設定される可能性が少ないと判断していたための条件不足。
- *No.3, No.4, No.13*
  - OSのオブジェクトに対して、自身への呼出し条件の不足。
- *No.6, No.7*
  - オーバランハンドラでの呼び出し条件不足。
- *No.14*
  - 可変長メモリブロック獲得判定処理で，タイムアウト判定時の条件不足。
- *No.12*
  - デバイス依存のサービスコールへの条件不足。
- *No.8-11*
  - ミューテックス部分で説明。

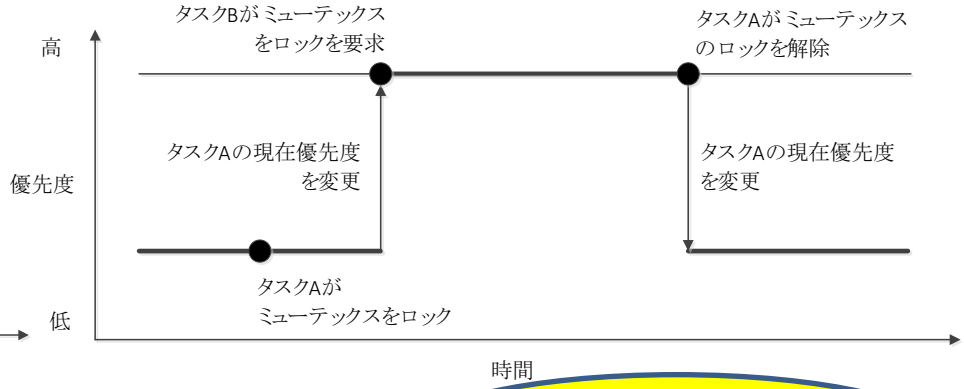
アジェンダ

1. はじめに
2. リアルタイムOSとは
3. リアルタイムOSのテスト設計
4. リアルタイムOSのテストで発見した問題と考察
5. ミューテックス部分で発見した問題と考察
6. まとめ

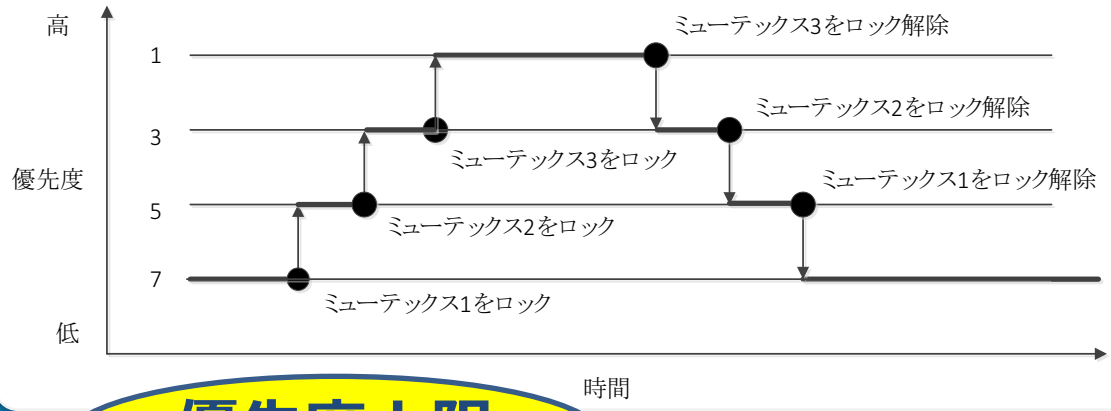
ミューテックス仕様について

- ミューテックス優先度逆転
  - ミューテックスには資源の排他制御を行う際に，排他する側とされる側のタスクの優先度の逆転（優先度の低いタスクが排他制御をすることによって優先度の高いタスクをロックしてしまう現象）を防ぐ「優先度上限プロトコル」と「優先度継承プロトコル」がある。





優先度継承



優先度上限

ミューテックスの属性は，タスクの待ち行列として「*FIFO*順」と「タスク優先度順」と上記がある

ミューテックスの問題の発見

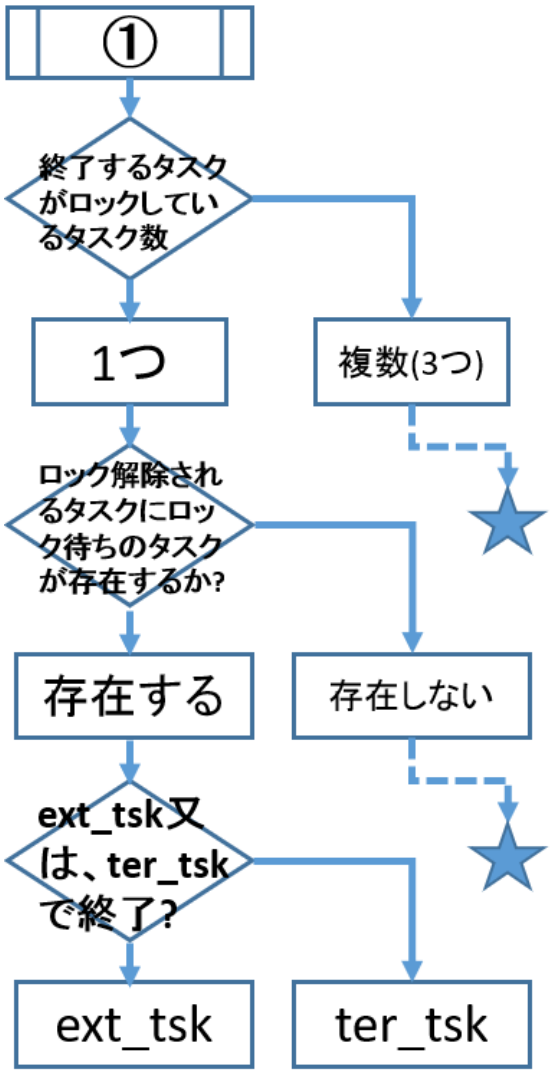
- ミューテックスでは，以下の部分で潜在的な問題を発見した。

① 複数のミューテックスをロックしているタスクを終了するテストケースを再検証する

仕様
- ミューテックスをロックしているタスクを終了された場合、そのロック状態を解除する仕組みがある

条件
- ①のようにテスト条件を検討
- 星印では，その左にある条件を繰り返し利用

結果
- 条件判定では全部では８通りとなる
- 不具合の発見と修正内容の検証が実施できた



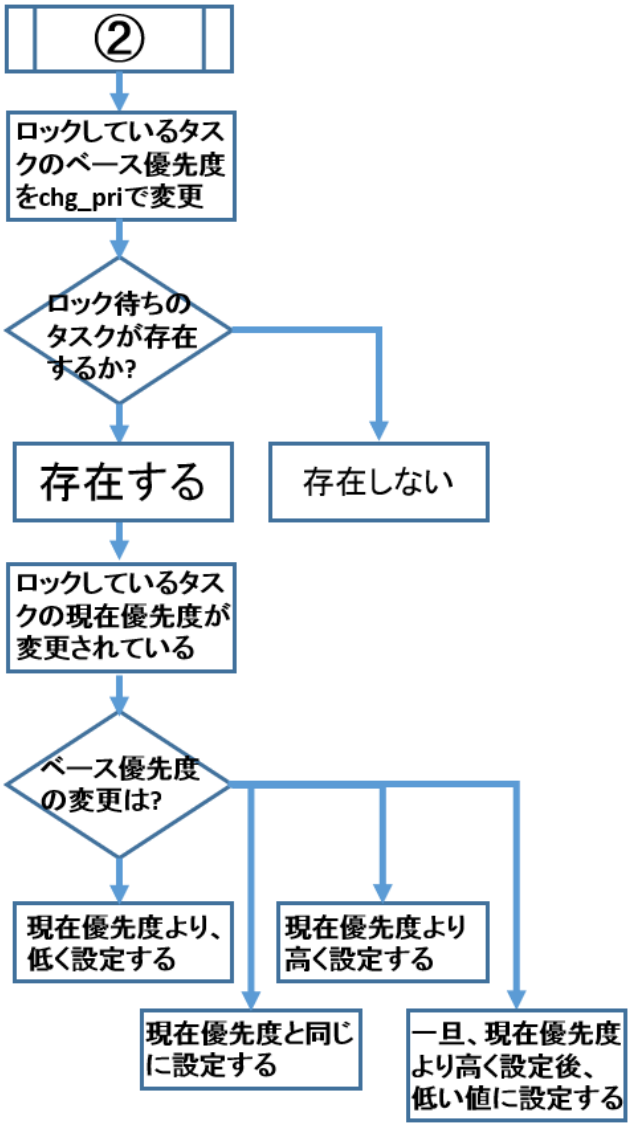
ミューテックスの問題の発見

- ミューテックスでは，以下の部分で潜在的な問題を発見した。
  - ② タスクの優先度を変化させる優先度上限・優先度継承のプロトコルの特徴に着目してテストケースを作成する

仕様
- 現在優先度と，ベース優先度がある
- 優先度継承では，現在優先度が変化する
- ロックが解除されると，ベース優先度に戻る

条件
- ロック解除の条件を*rel_wai*，*rel_mtx*，
- 複数のロック待ちの場合での現在優先度の処理などを検討する

結果
- 現在優先度の処理に問題を発見
- タイムアウトでの現在優先度の処理に問題を発見



アジェンダ

1. はじめに
2. リアルタイムOSとは
3. リアルタイムOSのテスト設計
4. リアルタイムOSのテストで発見した問題と考察
5. ミューテックス部分で発見した問題と考察
6. まとめ

※問題は最新のバージョンですべて解決済みで

まとめ

- リアルタイムOSでは，割込みでのイベント発生時に，サービスコールを処理する可能性がある。OSのスケジューリング処理は、アセンブラで処理が記述されており，このタイミングでの割込み発生の可能性もある。そのため，C言語のソースコードのカバレッジだけではリアルタイムOSのテストの網羅性を100%にすることはできない。

- 上記の前提条件から，リアルタイムOSのスケジューリングの考慮や，ユーザーの使用頻度や，不具合が発生したときの重大度の高いテストケースを想定してテストを実施することで，いくつかの不具合を発見し，修正を行うことが出来た。これは，テスト対象物の使用環境を考慮した上で想定されるリスクをできる限り小さくするリスクベースアプローチと言える。

- リスクベースアプローチを実施することにより，大規模あるいは複雑性が高いシステムにおいては，想定したより頻度が高いまたは被害の重大度が高いリスクを優先的に低減することで将来発生するインシデントの被害をより小さくすることにつながる。

参考文献

1. μITRON4.0仕様書，(社)トロン協会, 1999
2. μITRON4.0標準ガイドブック,トロン協会 (編集), 坂村 健 (監修),パーソナルメディア, 2001
3. ITRONプログラミング入門，金田一勤著， CQ出版社，2005年
4. 鴫原 一人, 一場 利幸, 本田 晋也, 高田 広章，“μITRONベースのマルチプロセッサ向けRTOSのテスト”情報処理学会論文誌 53巻12号，AN00116647，2682 – 2701pp，2012-12-15.